SUN-P9390

UNITED STATES PATENT APPLICATION FOR

A METHOD AND SYSTEM FOR IMPLEMENTING SUPER-USER-COMPATIBLE
PRIVILEGES

Inventor:

CASPER H. S. DIK

Prepared by:

WAGNER, MURABITO & HAO LLP

TWO NORTH MARKET STREET

THIRD FLOOR

SAN JOSE, CALIFORNIA 95113

(408) 938-9060

A METHOD AND SYSTEM FOR IMPLEMENTING SUPER-USER-COMPATIBLE
PRIVILEGES

BACKGROUND

5   The present invention relates to the field of computer operating system security policy implementation. Specifically, embodiments relate to providing a privilege framework in which traditional super-user based processes and traditional privilege based processes can coexist.

10   The traditional security policy model for Unix-based operating systems has only one mechanism for distinguishing between privileged processes and unprivileged processes. This mechanism is the association of all privileges to a user having a userid of zero, also known as a "super-user." A basic flaw of this model is the all-or-nothing approach. An application which needs a single
15   special privilege, such as a web server binding to a specific reserved port, a program running in the real-time scheduling class, a server to keep the clock synchronized, etc., all need to run or start with a userid of zero, e.g., as "root".

        In the traditional super-user model, a user with userid = 0 can read and
20   write to all files. In a traditional privilege model, one process may have the ability to read a file, but not to write to the file, for example. The privileges can be split into sub-privileges. In an operating system, one can typically have either super-user (userid = 0), as with traditional Unix operating environments, or traditional privileges.

25

There have been attempts by Unix vendors to combine the models, but to date, the typical model is based on allowing a system administrator to decide whether the system is to use the traditional super-user model (userid = 0) or the traditional privileges model or some combination of both.

5

SUMMARY

Thus, what is needed is a seamless method of extending privilege security policy, on a per-process basis, to processes designed to run with super-user security policy. What is further needed is a privilege policy for a system that allows traditional super-user based processes and traditional privilege based processes to coexist seamlessly and transparently.

The present invention provides, in various embodiments, a system for implementing a super-user-compatible privilege security policy on a computer operating system. The system includes a kernel for enforcing a security policy on processes, based on privileges. The system also includes a privilege model that interfaces with the kernel and implements a framework in which super-user based processes and privilege based processes transparently interface with the kernel. The privilege model includes several privilege sets associated with each process, a privilege awareness property state associated with each process, the property state indicating whether or not a process is privilege aware, and a software module for automatically modifying the privilege sets and the property state, on a per process basis, based on individual process behavior.

In one embodiment, a method of implementing a super-user-compatible privilege security policy on a computer operating system is disclosed. The method includes automatically extending the novel privilege security policy to super-user policy processes on a per-process basis, and, importantly, the super-user policy processes are unaware of the novel privilege security policy

extended to them. The method further includes automatically utilizing the novel privilege security policy for providing privileges for privilege policy processes on the computer operating system on a per-process basis. The privilege policy processes are, however, aware of the privilege security policy, which they are

5   utilizing.

Embodiments of the present invention provide a framework in which processes developed for super-user policies and processes developed for traditional privilege policies can transparently coexist within the same system

10   without modification to their underlying code.

In one embodiment, the super-user policy processes utilize a number of userids that include: a real userid that identifies a user; an effective userid that is utilized by a process for making access control decisions and that has a value

15   of either the real userid or of zero; and a saved userid that allows the effective userid to switch from the value of the real userid to the value of zero when the saved userid has a value of zero.

The privilege security policy processes, according to one embodiment of

20   the present invention, utilize a plurality of sets of privileges that includes: an inheritable set containing those privileges which are inherited when a second process overlays a first process; an effective set containing those privileges that are currently in effect in a process; a permitted set containing those privileges that a process is allowed to put into said effective set at will; and a limit set

containing an upper bound of privileges that a process and its sub-processes are allowed to obtain.

Importantly, the present invention also introduces a flag that indicates
5   whether or not the associated process is privilege aware (PA) or not privilege aware (NPA).

In accordance with one embodiment, a super-user policy process, running in a non-privilege-aware state and having an effective userid with a
10   value of zero, as observed by a security-policy-enforcement code, is extended by the effective set of privileges being equated to the limit set of privileges. Also, a super-user policy process, running in a non-privilege-aware state and having an effective userid with a value of zero, or a saved userid with a value of zero, or a real userid with a value of zero, as observed by a security-policy-
15   enforcement code, is extended by the permitted set of privileges being equated to the limit set of privileges.

A process, according to one embodiment of the present invention, transitions from a super-user policy process to a fully privileged process by
20   attaining a privilege-aware state. This can occur automatically by the process manipulating one or all of its privilege sets. A super-user based process having the effective userid equal to zero and transitioning from a non-privilege-aware state to the privilege-aware state does so by copying the limit set of privileges into both the effective set of privileges and the permitted set of privileges.

25

In one embodiment, a process transitions from the privilege-aware state to the non-privilege-aware state with no restrictions provided all userid's are non-zero. However, if any of the userid's equal zero, a process in the privilege-aware state can only transition to the non-privilege-aware state provided the

5   permitted set of privileges is equal to the limit set of privileges and, if the effective userid is equal to zero, the effective set off privileges is also equal to the limit set of privileges.

According to one embodiment of the present invention, a process may

10   directly manipulate its privileges. However, only a process in a privilege-aware state may manipulate the effective set of privileges. Removing privileges from the effective set is unrestricted, but adding privileges to the effective set of privileges is only allowable provided the privileges to be added are contained within the permitted set of privileges.

15

As discussed further below, the present invention introduces observed effective sets and observed permitted sets and implemented effective sets and implemented permitted sets on a process-by-process basis. The observed values are those actually used by the operating system in making privilege

20   decisions.

BRIEF DESCRIPTION OF THE DRAWINGS

The accompanying drawings, which are incorporated in and form a part of this specification, illustrate embodiments of the invention and, together with the description, serve to explain the principles of the invention:

5

FIGURE 1 is a block diagram of a single operating system within which super-user based processes and traditional privilege based processes can transparently coexist, in accordance with one embodiment of the present invention.

10

FIGURE 2 is a flow diagram illustrating an overview of changes in the privilege-aware state of processes, according to one embodiment of the present invention.

15　　　　FIGURE 3 is a flow diagram illustrating a super-user policy process operating in a non-privilege-aware state within a novel extended privilege policy system; according to one embodiment of the present invention.

FIGURE 4 is a flow diagram of a method of transitioning from a non-
20　　privilege-aware state to a privilege-aware state, according to one embodiment of the present invention.

FIGURE 5 is a flow diagram of a method of transitioning from a privilege-aware state to a non-privilege-aware state, according to one embodiment of the present invention.

25

FIGURE 6 is a flow diagram of a method of a process directly manipulating its privileges, in accordance with one embodiment of the present invention.

5

FIGURE 7 is a block diagram of an exemplary computer system used in accordance with one embodiment of the present invention.

DETAILED DESCRIPTION

Reference will now be made in detail to embodiments of the invention, examples of which are illustrated in the accompanying drawings. While the invention will be described in conjunction with embodiments, it will be

5   understood that they are not intended to limit the invention to these embodiments. On the contrary, the invention is intended to cover alternatives, modifications and equivalents, which may be included within the spirit and scope of the invention as defined by the appended claims. Furthermore, in the following detailed description, numerous specific details are set forth in order to

10   provide a thorough understanding of the present invention through a discussion of embodiments thereof. However, the present invention may be practiced without these specific details. In other instances, well-known methods, procedures, components, and circuits have not been described in detail as not to unnecessarily obscure aspects of the present invention.

15

Some portions of the detailed descriptions, which follow, are presented in terms of procedures, logic blocks, processing, and other symbolic representations of operations on data bits within an electronic computing device and/or memory system. These descriptions and representations are the means

20   used by those skilled in the data processing arts to most effectively convey the substance of their work to others skilled in the art. A procedure, logic block, process, etc., is herein, and generally, conceived to be a self-consistent sequence of steps or instructions leading to a desired result. The steps are those requiring physical manipulations of physical quantities. Usually, though

25   not necessarily, these physical manipulations take the form of electrical or

magnetic signals capable of being stored, transferred, combined, compared, and otherwise manipulated in a computer system or similar electronic computing device. For reasons of convenience, and with reference to common usage, these signals are referred to as bits, values, elements, symbols,

5   characters, terms, numbers, or the like with reference to the present invention.

It should be borne in mind, however, that all of these terms are to be interpreted as referencing physical manipulations and quantities and are merely convenient labels and are to be interpreted further in view of terms

10   commonly used in the art. Unless specifically stated otherwise as apparent from the following discussions, it is understood that throughout discussions of the present invention, discussions utilizing terms such as "receiving", "processing", "creating", "storing", "copying", "accessing", "generating", "providing", "performing", "manipulating", "sending", "replacing", "gaining",

15   "attaining", "transitioning", or the like, refer to the action and processes of a computer system, or similar electronic computing device, that manipulates and transforms data. The data is represented as physical (electronic) quantities within the computing device's registers and memories and is transformed into other data similarly represented as physical quantities within the computing

20   device's memories or registers or other such information storage, transmission, or display devices.

In one embodiment of the present invention, a method of implementing a super-user-compatible privilege security policy on a computer operating system

25   is herein disclosed. The method includes providing a framework in which

super-user based processes are automatically and transparently extended the privilege security policy and can coexist with traditional privilege based processes within a single operating system. The super-user based processes may operate with a lack of awareness of the privilege security policy that is
5   extended to them, while the traditional privilege policy processes are fully aware of the privilege security policy, which they are utilizing.

Certain portions of the detailed descriptions of embodiments of the invention, which follow, are presented in terms of processes and methods.
10   Although specific steps are disclosed herein describing the operations of these processes and methods, such steps are exemplary. That is, embodiments of the present invention are well suited to performing various other steps or variations of the steps recited in the flowcharts of the figures herein.

15   In accordance with at least one embodiment of the present invention, the security policy model that is evolved is one with which ordinary users can have additional privileges and a user with a userid of zero is no longer all-powerful. For purposes of brevity and clarity, this security policy model will be discussed in the context of a Unix-based operating environment. It should be understood,
20   however, that the present invention is applicable to any operating environment that operates with a privilege security model.

Referring now to Figure 1, a block diagram of a single operating system 100 within which super-user based processes and traditional privilege based
25   processes can transparently coexist, is presented in accordance with one

embodiment of the present invention. Privilege policy decision maker 110 is a kernel process that enforces the rules that govern the extension and modification of privileges. These privileges are extended to processes operating within novel extended privilege policy 120. Novel extended privilege policy 120 can modify

5   allowed privilege sets and a privilege awareness property state, based on the behavior of the individual process. The extension of privileges to processes is performed on a process-by-process basis. Super-user based process 130 and traditional privilege process 140 are representative of a number of processes, having either super-user based security policy or traditional privilege security

10   policy, that can coexist and execute within the novel extended privilege policy domain while remaining transparent to one another. In addition, they can execute while remaining unaware of privileges that have been extended to them and they can transition between a state of awareness of their privileges, a "privilege-aware" state and a state of unawareness of their privileges, a "non-

15   privilege-aware" state.

Embodiments of the present invention provide a framework in which processes developed for super-user policies and processes developed for traditional privilege policies can transparently coexist within the same system

20   without modification to their underlying code. In one embodiment of the present invention, the security policy model is a privilege-awareness model. The privilege state of a process is extended with a Privilege Aware State (pas) that can take the values PA (privilege-aware) and NPA (non-privilege-aware). This pas is in the form of a property that may be set to either PA or NPA by a process

when certain conditions are met. These conditions are discussed in detail in conjunction with Figures 2-5 below.

According to one embodiment of the present invention, a process
5 running with a privilege security policy has four sets of privileges: an inheritable set (I) that contains those privileges which are inherited when one process image overlays another process image; an effective set (E) that contains those privileges that are currently in effect in a process; a permitted set (P) that contains those privileges that a process is allowed to put into the effective set at
10 will; and a limit set (L) that contains an upper bound of privileges that a process and its sub-processes are allowed to obtain or inherit. L has been added in accordance with one embodiment of the present invention. L is only enforced at the time a new process image overlays the current process image, allowing a process to drop privileges at that time, while still using them until that point in
15 time.

A process running with a super-user security policy model carries with it the following userids: a real userid, an effective userid and a saved userid. The real userid identifies the specific user; the effective userid is used by a process
20 for making access control decisions. The effective userid may have a value of zero, allowing it unlimited access, or the value of the real userid, allowing it only access to the specific user's files. The saved userid allows the effective userid to toggle between the real userid and zero when the saved userid has attained a value of zero. Otherwise, if the saved userid is not zero, the effective userid
25 maintains the value of the real userid.

A concept of observed Effective and Permitted sets, E^o and P^o, are used to describe the E and P as observed by the code that enforces the operating system security policy, the process itself and other processes, according to one

5 embodiment of the present invention. The implementation Effective and Permitted sets, E^i and P^i, are the sets which are actually stored in the process credential.

Equations 1 and 2 below summarize the way an observer (e.g., privilege

10 policy decision maker 110 of Figure 1) sees E and P behave in terms of the credential (C) held by the process, the credential indicating the qualifications the process possesses for obtaining access to a set of privileges:

(1) $\quad$ $C.E^o = C.e \neq 0 \vee C.pas = PA\ ?\ C.E^i : C.L$

15 (2) $\quad$ $C.P^o = (C.e \neq 0 \wedge C.s \neq 0 \wedge C.r \neq 0) \vee C.pas = PA\ ?\ C.P^i : C.L$

where: C is the process credential; E^o is the observed effective set ($C.E^o$ thus is the process credential of the observed effective set); e is the effective userid; pas is the privilege awareness state; PA is the state of being Privilege Aware; E^i is the implementation effective set; L is the limit set; P^o is the observed

20 permitted set; s is the saved userid; r is the real userid; and P^i is the implementation permitted set.

In equation (1), according to one embodiment of the present invention, an observer sees the credentials of the effective set as the same as that of the

implementation effective set provided either the effective userid is not equal to zero or the privilege awareness state is PA. Otherwise, the value of E^o is set to that of the limit set, L. An alternate way of stating this observation rule is that a super-user policy process running in an NPA state and having a userid of zero,

5 as observed by a security-policy-enforcement code, is extended the effective set of privileges that is the limit set. Otherwise, E^o is equal to E^i.

In equation (2) above, an observer sees the credentials of the permitted set as those of P^i, provided either none of the userids are equal to zero or the

10 privilege awareness state is PA, according to one embodiment of the present invention. Otherwise, the value of P^o is set to L. An alternate version of this observation rule is that a super-user policy process, running in a non-privilege-aware state and having any of its userids equal to zero, as observed by a security-policy-enforcement code, is extended the permitted set of privileges that

15 is the limit set. Otherwise, P^o is equal to P^i.

In one embodiment, the initial process starts off with E^i equal to P^i which is equal to I which is equal to I^o and L equal to all privileges. I^o is the set of basic privileges which can be empty. This default process credential behaves exactly

20 like the super-user model. That is, when the effective userid is non-zero, the process has the default privileges; when the effective userid is zero, the process has all privileges.

Embodiments of the present invention provide a framework in which

25 processes developed for super-user policies and processes developed for

traditional privilege policies can transparently coexist within the same system without modification to their underlying code. Therefore, when processes developed for super-user policies need to start programs developed for traditional privilege policies, and vice versa, it can become necessary for a

5   process to change its privilege awareness state (pas) to one compatible with the process with which it is interacting.

Figure 2 is a flow diagram 200 illustrating an overview of changes in the privilege aware state (pas) of processes, according to one embodiment of the

10   present invention. The pas is a state variable that is set to indicate the privilege aware state to the privilege policy decision maker (e.g., 110 of Figure 1). Processes can change their pas. Changing pas is mostly automatic and restricted. As shown in step 205, a process that is operating unaware of privileges has a pas of NPA (not privilege aware).

15

Still referring to Figure 2, at step 210, a process operating in an NPA state can directly change its state to that of privilege aware (PA) if it becomes necessary for an operation, according to one embodiment of the present invention. This is shown by the path through step 215 that ends at step 230 with

20   the privilege aware state variable changed to PA.

In step 220 of Figure 2, in one embodiment, when an NPA process manipulates privileges (e.g., E or P) it becomes privilege aware automatically and pas becomes PA, as shown in step 230. Otherwise, it continues to operate as a

25   NPA process. If it does manipulate E or P, then the observed privilege sets, E^o

and P° become decoupled from the Limit set, L. Similarly, if L is changed, then E° and P° must remain unchanged which can only be achieved by a change to PA (see equations 1 and 2 above).

5   There are some situations in which a process may need to lose its privilege aware state (e.g., when a copy of an application is being overlaid with another application, such as upon an "exec(2)" Unix system call). In step 240 of Figure 2, if a PA process needs to become NPA in order to perform an operation, it may change its state variable only if it meets certain restrictions, as shown by

10 step 250. The change from PA to NPA is not automatic. In step 250, a PA process can only become NPA if it meets certain restrictions. For example, a PA process can only become NPA if E and P can be adjusted in a manner that keeps E° and P° unchanged and when such a change does not lead to the process leveraging a userid of zero into full privileges. Provided these restrictions are

15 met, then a PA process is allowed to change the process state variable and transition to NPA as shown by the path to step 210. Otherwise, step 260 is entered and the transition is not allowed, and thus the process state variable remains set to PA as shown by the path to step 230.

20   In order to understand the operation of a super-user policy process in a novel extended privilege policy system, Figure 3 is presented. Figure 3 is a flow diagram illustrating a super-user policy process 300 operating in a non-privilege-aware (NPA) state within a novel extended privilege policy system, according to one embodiment of the present invention. In this process, privilege

25 policy is extended to super-user policy processes on a per-process basis. In

step 310, a super-user policy process begins execution in a NPA state. In accordance with one embodiment of the present invention, when a super-user policy process executes, the operating system will recognize it as a super-user based process and will automatically extend it privileges based on the

5   configuration of its userids (user identifications).

In step 320 of Figure 3, in accordance with one embodiment of the present invention, a check is performed as to whether any of the userids are equal to zero. If so, the method moves to step 330. If not, method 300 moves to step 340.

10   In step 330, since there is at least one userid equal to zero, the permitted set (C.P) that is being observed becomes equal to the limit set (C.L), thus allowing all privileges in the limit set to be permitted. However, in step 340, if no userids were zero, the permitted set (C.P) retains the value of the implementation permitted set. This process is performed by the novel extended privilege policy (e.g., 120 of

15   Figure 1) in conjunction with privilege policy decision maker (110 of Figure 1), without the super-user process being aware.

In step 350 of method 300, if a super-user policy process has an effective userid of zero, the effective set (C.E) that is being observed by the system's

20   security policy becomes equal to the limit set (C.L) as shown in step 360, in accordance with one embodiment of the present invention. Thus, a super-user based process having its effective userid equal to zero is extended the limit set (C.L) of privileges for both the effective set (C.E) and the permitted set (C.P).

Still referring to Figure 3, in step 370 of method 300, according to one embodiment of the present invention, if the effective userid is not equal to zero, the effective set (C.E) retains the implementation effective set of privileges. At this point the process continues to execute as always, unaware of the privilege sets

5   that have been allocated, and process 300 is exited.

In summary of Figure 3, according to one embodiment of the present invention, a process running in an NPA state behaves almost exactly like a traditional super-user based process. It may have privileges in E and P, and the

10 operating system will honor those, but if the process's userid becomes set to zero, the upper bound of that process's privileges, L, will be observed to replace whatever value is in E. Similarly, if any process's userid becomes zero, L is used as P^o. The observer will see E^o follow the transition of the effective userid and it will see P^o revert when all of the process's userids are no longer zero. In contrast,

15 the implementation sets E^i and P^i remain unchanged during userid manipulations.

To further understand how a process changes from running in a non-privilege aware state to becoming privilege-aware, Figure 4 is presented. Figure

20 4 is a flow diagram of a method 400 of transitioning from a non-privilege-aware state to a privilege-aware state, according to one embodiment of the present invention. Basically, there are no restrictions on becoming PA. In step 410 of Figure 4, a super-user based process is running in a NPA state and needs to transition to PA. The following steps indicate the changes to the process

25 credentials, which occur.

In step 420 of Figure 4, in accordance with one embodiment of the present invention, a check is performed as to whether any of the userids are equal to zero. If so, the method moves to step 430. If not, method 400 moves to step 440.

5 In step 430, since there is at least one userid equal to zero, the permitted set that is being implemented becomes equal to the limit set, thus allowing all privileges to be permitted. However, in step 440, if no userids were zero, the permitted set retains the value of the default implementation set.

10 In step 450 of method 400, if a super-user policy process transitioning from NPA to PA has an effective userid of zero, the privileges of the limit set are copied into the effective set to be implemented as shown in step 460 in accordance with one embodiment of the present invention. Thus, a super-user based process having its effective userid equal to zero and transitioning from a non-privilege-

15 aware state to a privilege aware state copies the limit set of privileges into to both the effective set and the permitted set.

Still referring to Figure 4, in step 470 of method 400, according to one embodiment of the present invention, if the effective userid is not equal to zero,

20 the effective set retains the default implementation effective set of privileges. In step 480, the previously NPA process now runs in a fully privilege-aware state and, thus, can manipulate privileges directly. The observed effective and permitted sets (E^o and P^o.) remain unchanged. At this point userid manipulations no longer effect on the privilege status and method 400 is exited.

25

To more thoroughly explain the transitioning of a process from a privilege aware state to a non-privilege aware state, Figure 5 is presented. Figure 5 is a flow diagram of a method 500 of transitioning from a privilege-aware state to a non-privilege-aware state, according to one embodiment of the present invention.

5   There are some situations in which a process may need to lose its privilege aware state (e.g., when a copy of an application is being overlaid with another application, such as upon an "exec(2)" system call). Several restrictions apply when a PA state transitions to an NPA state, as not all combinations of privilege sets can be successfully and safely converted to an equivalent NPA process.

10

In step 510 of method 500, according to one embodiment, a process in a privilege-aware state wishes to transition to a non-privilege-aware state. At step 520, a check is performed to see if any userids are set to zero. If none of the userids (effective, real or saved) are equal to zero, then there are no restrictions

15   and the transition is permitted and method 500 moves to step 560. However, if any of the userids are equal to zero, there are restrictions that apply and the other steps must be followed.

In step 530 of Figure 5, if the implementation permitted set, P^i is the same

20   as the limit set, then there are no further restrictions, according to one embodiment of the present invention, and method 500 moves to step 540. However, if P^i is not equal to the limit set, then method 500 moves to step 555 and the transition to NPA is not permitted.

In step 540, provided P^l equals the limit set, the effective userid is interrogated. In one embodiment of the present invention, if it is found that the effective userid is not equal to zero, the transition is permitted and method 500 moves to step 560. However, if the effective userid is equal to zero, then the

5 implementation effective set, E^l must also be equal to the limit set as shown in step 550 of Figure 5.

Still referring to Figure 5, if the implementation effective set E^l is not equal to the limit set, L, then, according to one embodiment of the present

10 invention, the transition to NPA is not allowed and method 500 moves to step 555 and then is exited. Otherwise, the transition is allowed and step 560 is entered. In step 560, the process state is now NPA. Either no userids are equal to zero or, if any of the userids are equal to zero, the implementation permitted set is granted only those privileges in the intersection of the limit set and the

15 inheritable set. Also, if the effective userid is equal to zero, the implementation effective set is granted only those privileges in the intersection of the limit set and the inheritable set. It should be understood that, under these operations, the observed sets, E^o and P^o, remain unchanged.

20 There are various times, according to one embodiment of the present invention, at which the per-process state changes, either as a result of explicit actions, or as a side effect of other actions. When userids change the changes are only observed and are not actually reflected in the underlying data structures.

25

In all circumstances, according to one embodiment, the actual implementation sets remain unchanged. In PA processes, when userids change, no changes are observed by the observed sets E^o and P^o. In processes having no userids set to zero, no changes are observed, even if

5   privileges are used to obtain a userid of zero. Then, in NPA processes, the E^o will alternate between L and the E^i, depending on the value of the effective userid, and P^o will appear as L until the last userid is set to non-zero, in which case it will revert to the privileges of P^i.

10   Figure 6 is a flow diagram of a method 600 for a process to directly manipulate its privileges, changing its effective set, in accordance with one embodiment of the present invention. In step 610 of method 600, a process wishes to manipulate its privileges directly, changing the credentials of its effective set. If the process is PA, as examined at step 620, then method 600

15   moves to step 630. If the process is NPA then method 600 moves to step 625 and the process transitions to PA using the method described in Figure 2. Once the process becomes PA, then it moves to step 630.

At step 630 of Figure 6, the process is interrogated, according to one

20   embodiment of the present invention, regarding whether it is adding privileges to the effective set. If privileges are being added to E, then method 600 moves to step 640. If privileges are not being added to the effective set, then, at step 635, the process is interrogated as to whether it is removing privileges from P. If the process is not shrinking P, a check is performed at step 660 as to whether it

25   is increasing P. If not, then there are no restrictions and method 600 moves to

step 680 and method 600 is then exited. If it is wishing to increase P, then method 600 proceeds to step 670 and the change is not permitted.

If, at step 635, privileges were being removed from P, then method 600

5   moves to step 650 where it removes the same privileges from E, in accordance with one embodiment of the present invention. Then method 600 may move to step 680 and the privilege change is permitted, after which method 600 is exited.

10   In one embodiment, if privileges are being added to the credentials of E at step 630, then the permitted set, P, is interrogated at step 640 for the privileges to be added. If the privileges to be added are in P, then method 600 proceeds to step 680 where the change is permitted and method 600 is then exited. However, if the privileges to be added are not in P, then method 600

15   proceeds to step 670 and the change is not permitted. Method 600 is then exited.

The foregoing discussion with respect to Figure 6 treated changing the effective set of privileges according to one embodiment of the present invention. It should be understood that other privilege sets can also be directly

20   manipulated. Removing privileges from E is not restricted, but only privileges in P may be added to E. A process manipulating E needs to be PA, and if it is NPA, it will transition to PA following the method described in Figure 2. P can only be shrunk. A process so designed will shrink P if it has no future need for

certain privileges in P. Privileges removed from P are automatically removed from E and NPA processes manipulating P will automatically transition to PA.

In one embodiment, the inheritable set, I, may be changed. Privileges
5 from P may be added freely to I and privileges may be removed from I without restriction. Since changing I does not influence E^o and P^o, an NPA process does not need to transition to PA when manipulating I. Removing privileges from P does not effect I (e.g., I is allowed to be a superset of P).

10 According to one embodiment of the present invention, privileges may not be added to the limit set by any mechanism. Privileges may be removed from the limit set, but with extreme caution as applications running with userid equal zero usually assume full privileges on the operating system. This often leads to no error checking being performed for privileged operations. Changing
15 the limit set influences E^o and P^o and, as a result, an NPA process needs to transition to PA when manipulating L.

According to one embodiment of the present invention, a process that manipulates its privileges is so marked in order to award the process additional protection. It should also be noted that, in another embodiment, using certain
20 commands (e.g., "getpflags" and "setpflags") processes may query and change the privilege aware state of a process.

With reference now to Figure 7, a block diagram of an embodiment of an
25 exemplary computer system 700, used in accordance with the present

invention, is illustrated. It should be appreciated that system 700 is not strictly limited to be a computer system. As such, system 700 of the present embodiment is well suited to be any type of computing device (e.g., server computer, portable computing device, desktop computer, etc.). Within the

5   following discussions of the present invention, certain processes and steps are discussed that are realized, in one embodiment, as a series of instructions (e.g., software program) that reside within computer readable memory units of computer system 700 and executed by a processor(s) of system 700. When executed, the instructions cause computer 700 to perform specific actions and

10   exhibit specific behavior that is described in detail herein.

     Computer system 700 of Figure 7 comprises an address/data bus 710 for communicating information, one or more central processors 702, e.g., privilege policy decision maker 110, coupled with bus 710 for processing information and

15   instructions. Central processor unit(s) 702 may be a microprocessor or any other type of processor. The computer 700 also includes data storage features such as a computer usable volatile memory unit 704 (e.g., random access memory, static RAM, dynamic RAM, etc.) coupled with bus 710 for storing information and instructions for central processor(s) 702, a computer usable

20   non-volatile memory unit 706 (e.g., read only memory, programmable ROM, flash memory, EPROM, EEPROM, etc.) coupled with bus 710 for storing static information and instructions for processor(s) 702. System 700 also includes one or more signal generating and receiving devices 708 coupled with bus 710 for enabling system 700 to interface with other electronic devices and computer

systems. The communication interface(s) 708 of the present embodiment may include wired and/or wireless communication technology.

Optionally, computer system 700 may include an alphanumeric input

5   device 714 including alphanumeric and function keys coupled to the bus 710 for communicating information and command selections to the central processor(s) 702. The computer 700 can include an optional cursor control or cursor directing device 716 coupled to the bus 710 for communicating user input information and command selections to the central processor(s) 702.

10

The system 700 of Figure 7 may also include one or more optional computer usable data storage devices 718 such as a magnetic or optical disk and disk drive (e.g., hard drive or floppy diskette) coupled with bus 710 for storing information and instructions. An optional display device 712 is coupled

15   to bus 710 of system 700 for displaying video and/or graphics. It should be appreciated that optional display device 712 may be a cathode ray tube (CRT), flat panel liquid crystal display (LCD), field emission display (FED), plasma display or any other display device suitable for displaying video and/or graphic images and alphanumeric characters recognizable to a user.

20

Thus, the present invention provides, in various embodiments, a method and system for implementing super-user-compatible privileges. The foregoing descriptions of specific embodiments have been presented for purposes of illustration and description. They are not intended to be exhaustive or to limit

25   the invention to the precise forms disclosed, and many modifications and

variations are possible in light of the above teaching. The embodiments were chosen and described in order to best explain the principles of the invention and its practical application, to thereby enable others skilled in the art to best utilize the invention and various embodiments with various modifications as are

5   suited to the particular use contemplated. It is intended that the scope of the invention be defined by the claims appended hereto and their equivalents.